SoK: Lifting Transformations for Simulation
Extractable Subversion and Updatable SNARKs

Behzad Abdolmaleki, ¹ Sebastian Ramacher, and Daniel Slamanig ²

¹ University of Tartu, Estonia
² AIT Austrian Institute of Technology, Austria

23.04.2020/ The 3rd ZKPROOF

The full version of this work will appear in ACM CCS 2020
Outline

NIZK, SNARK, subversion and updatable setup

COC0 Framework [KZM+15]

OCOC0 (optimized version of COC0)

New framework: Lamassu

SE Subversion SNARK

SE Subversion SNARK

SE updatable SNARK
NIZK in CRS model

Correctness
Soundness
Zero-knowledge

\[ x \in L, w \]

\[ \pi = P(\text{crs}, x, w) \]

\[ \pi \approx \pi^* \]

\[ \pi^* = \text{Sim}(\text{td}, x) \]

\[ V(\text{crs}, x, \pi) ? \]

\[ V(\text{crs}, x, \pi) ? \]

\[ V(\text{crs}, x, \pi) ? \]
Special Type of NIZKs: zk-SNARKs

Witness \( w \) \((x, w) \in \mathbb{R}_L\)

\((\text{CRS}, \text{td}) \leftarrow \text{CRSGen}(1^n)\)

proof \(\leftarrow\) \text{Prove}(\text{CRS}, \text{witness})

\(\{1, 0\} \leftarrow \text{Verify}(\text{CRS}, \text{word}, \text{proof})\)

OK, \( x \in L\)

We are interested to make them succinct as much as possible.

**Succinct:** Small size of proof and low verification complexity


Proof = 3 group elements
Subversion security: Impossibility Result

- [Bellare, Fuchsbauer, Scafuro 2016]

<table>
<thead>
<tr>
<th></th>
<th>WI</th>
<th>ZK</th>
<th>Sub-ZK</th>
</tr>
</thead>
<tbody>
<tr>
<td>SND</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Sub-SND</td>
<td>Possible</td>
<td>Impossible</td>
<td>Impossible</td>
</tr>
</tbody>
</table>
SNARKs in Different Subversion Setting

- **SNARK in the CRS Model** (i.e., [EC:Groth16])
- **Subversion Zero-Knowledge SNARK** (i.e., [AC:ABLZ17] [PKC:Fuc18])
- **Updatable SNARK** (i.e., [C:GKM+18]...)

Diagram: 
- P → CRS
- V → CRS
- CRS → Proof
- CRS → Proof
- CRS → Proof
Achieving Simulation Extractability
Soundness to **Knowledge Soundness**

I) **Blackbox way**: Folklore compiler [SP92]: “x ∈ L and I have encrypted w”

- Ω: perfectly correct IND-CPA public-key encryption

- Extend CRS with a **public key of Ω : pk**

- Extend proof with encryption of w: c = Enc(pk; w, r)

- Extend statement to (x, w) ∈ R_L ∧ c = Enc(pk; w, r)

- Put secret key of Ω in extraction trapdoor t_{d_{ext}} ~ w ← (t_{d_{ext}}, c)

II) **Non-Blackbox way**: The original statement (x, w) ∈ R_L remains unchanged but the knowledge soundness is under some non-falsifiable assumption (some knowledge assumptions), i.e., SNARKs constructions…
On Simulation Soundness

In a real world protocol:

• Adversary sees many different proofs

• Might be possible to turn proof for word $x$ into a proof $\pi' \neq \pi$

• Or worse: turn into a proof $\pi'$ for a different word $x' \neq x$

Hence

• Adversary may query proofs
• Must produce a proof not queried before

Similar issue for signatures: one-time EUF-CMA – EUF-CMA – strong EUF-CMA
Soundness to Simulation Soundness

Folklore compiler [GMY03; Gro06]:
“\(x \in L\) or I know a signature under a public key in the CRS”

- \(\Sigma\): EUF-CMA signature scheme
- \(\Sigma^1\): strong one-time signature scheme

- Extend CRS with a public key of \(\Sigma : pk_\Sigma\)

- For a proof
  - Generate new key pair for \(\Sigma^1\)
  - Extend statement to
    \[(x, w) \in R_L \vee \text{Verify}(pk_\Sigma, pk_{\Sigma^1}, \sigma) = 1\]
  - Sign proof with \(sk_{\Sigma^1}\)
  - Put \(sk_\Sigma\) secret key of \(\Sigma\) in simulation trapdoor \(td\)
Composable 0-knowledge, Compact 0-knowledge (COCO) Framework

\[ [KZM+15] \]

- Extend statement to
  \[ c = Enc(pk_\Omega; w, r_1) \land (x; w) \in R_L \lor \mu = f_s(pk_{\Sigma^1}) \land \rho = Commit(s, r_0) \]

- and sign \((x, c, \mu, \pi_L)\) with \(sk_1\)

- The crs extended with \((\rho, pk_{\Sigma})\) and with \((s, r_0)\) simulation trapdoor, \(sk_\Omega\) extraction trapdoor

- \(\Omega\): public-key encryption
- \(\Sigma^1\): strong one-time signature
- \(f\): PRF
- \(Commit\): Commitment
Composable 0-knowledge, Compact 0-knowledge (COCO) Framework [KZM+15]

- Extend statement to
  \[ c =: Enc(pk_\Omega; w, r_1) \land \left( (x; w) \in R_L \lor \mu = f_s(pk_{\Sigma^1}) \land \rho = \text{Commit}(s, r_0) \right) \]
  
- and sign \((x, c, \mu, \pi_L)\) with \(sk_1\)

- The crs extended with \((\rho, pk_{\Sigma})\) and with \(\Omega\)

- \(\Omega\): public-key encryption
- \(\Sigma^1\): strong one-time signature
- \(f\): PRF
- \(\text{Commit}\): Commitment

Alternatives: Boneh-Boyen [BB04], Groth sOTS [G06]

\(- f: \text{SHA256} \)
\(- \text{Commit}: \text{SHA256} \)
Proving pre-image of a random oracle

How to commit to the PRF key while retaining provable security?
Result I: The OC0C0 Framework
[ARS20]
The OC0C0 Framework [ARS20]

- Fixed-value key-binding PRF [CMR98; Fis99]

For a PRF $f$ with key $s$ and special value $\beta$, hard to find $s_0$ with $f_s(\beta) = f_{s_0}(\beta)$

- Change statement to

$$(x; w) \in R_L \lor \mu = f_s(p\kappa_1) \land \rho = f_s(\beta)$$
Number of constraints required for C0C0 and OC0C0

<table>
<thead>
<tr>
<th>Framework Symmetric primitive</th>
<th>PRF / Commitment</th>
<th>Provably secure</th>
<th># of constraints PRF / Com.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0C0</td>
<td>HMAC PRF + hash com.</td>
<td>✗</td>
<td>111360 + 44544 244992</td>
</tr>
<tr>
<td>SHA256</td>
<td>HMAC PRF</td>
<td>✓</td>
<td>111360 222720</td>
</tr>
<tr>
<td>SHAKE256</td>
<td>TLS 1.2 PRF</td>
<td>✓</td>
<td>230400 460800</td>
</tr>
<tr>
<td>MiMC-(1025, 646)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>38400 76800</td>
</tr>
<tr>
<td>GMiMC-(1024, 4, 332)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>646 1292</td>
</tr>
<tr>
<td>Poseidon-(1536, 2, 10, 114)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>999 1998</td>
</tr>
<tr>
<td>Vision-(1778, 14, 10)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>402 804</td>
</tr>
<tr>
<td>Rescue-(1750, 14, 10)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>1400 2800</td>
</tr>
<tr>
<td>LowMC-(256, 256, 1, 537)</td>
<td>feed-forward PRF</td>
<td>✓</td>
<td>840 1680</td>
</tr>
<tr>
<td>LowMC-(1024, 256, 1, 1027)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>1074 2148</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2144 4288</td>
</tr>
</tbody>
</table>
Result II: Lamassu (framework for subversion and updatable) [ARS20]
The Tree of Succinct NIZK

NIZK in CRS Model

zk-SNARKs

- **Updatable** zk-SNARKs [GKM+18]
- **Sub**- zk-SNARKs [ABLZ17] [Fuc17]

**Lamassu:**
- zk-SNARKs
- **Sub** zk-SNARKs
- **Updatable** zk-SNARKs
Lamassu [ARS20]

- Generic framework to obtain:
  simulation extractable (SE) subversion or updatable zk-SNARKs

- Built from
  - updatable signatures
  - alternative compiler for SE [DS19]
Key-homomorphic Signatures

- Homomorphism between secret-key and public-key spaces: $\mu: S \rightarrow P$
  Natural in the DLOG setting: $x \rightarrow g^x$

- Signatures can be adapted from $pk$ to $pk_0 = pk \cdot \mu(sk_0 - sk)$ if $(sk_0 - sk)$ known

- Examples: Schnorr, BLS, and many more
SE property using Key-Homomorphic Signatures

- Compiler [DS19]:
  - $\Sigma$: key-homomorphic EUF-CMA signature scheme
  - $\Sigma^1$: one-time signature scheme
  - Extend CRS with a public key of $\Sigma$: $pk$

For a proof
- Generate key pairs $(sk', pk')$ for $\Sigma$ and $(sk^1, pk^1)$ for $\Sigma^1$
- Extend statement to
  $(x; w) \in R_L \forall pk' = pk. \mu(sk' - sk_\Sigma)$

- Sign $pk^1$ with $sk'$ and sign the proof with $sk^1$

Put secret key $sk_\Sigma$ of $\Sigma$ in simulation trapdoor $td_{sim}$ Sim

- Obtain SE subversion zk-SNARK
Updatable Signatures

- Similar to updatable CRS
  - $U_{pk}$: update $pk$ and provide proof of update
  - $V_{pk}$: verify update

- Idea: either original $pk$ created honestly or update was done honestly
  - Example: Schnorr in bilinear groups with BDH knowledge assumption

$$(g_1^x, g_2^x) \leftarrow (g_1, g_2; r)$$

$$x \leftarrow (g_1, g_2; r)$$
Framework for SE (Sub & Updatable) zk-SNARK

LAMASSU

\[ L' = \{ (x, w) \in R_L \land pk' = pk \cdot \mu(sk' - sk) \} \]
\[ x^\text{new} = (x, pk'), w^\text{new} = (w, sk' - sk) \]
\[ \text{CRS}^\text{new} = (crs_{\text{snark}}, pk) \]

Updatable Signature

SE zk-SNARK
SE sub zk-SNARK
SE updatable zk-SNARK
## Comparison of SE SNARKs

<table>
<thead>
<tr>
<th>Features</th>
<th>crs</th>
<th>Overhead</th>
<th>π bits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>bits</td>
<td></td>
</tr>
<tr>
<td>generic subversion updatable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C∅C∅ [KZM+15]†</td>
<td>✓</td>
<td>✓*</td>
<td>x</td>
</tr>
<tr>
<td>C∅C∅[S]</td>
<td>✓</td>
<td>✓*</td>
<td>x</td>
</tr>
<tr>
<td>C∅C∅[G]</td>
<td>✓</td>
<td>✓*</td>
<td>x</td>
</tr>
<tr>
<td>LAMASSU[S,S]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>LAMASSU[S,S]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LAMASSU[S,G]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>LAMASSU[S,G]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LAMASSU[S,BB]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>LAMASSU[S,BB]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Groth-Maller [GM17]</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bowe-Gabizon [BG18]</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Lipmaa (S_{s}^{se}) [Lip19]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Kim-Lee-Oh [KLO19]</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Atapoor-Bagheri [AB19]†</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bagheri [Bag19]†</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

† Proves statements with respect to the evaluation of a random oracle (cf. Section 3.2).
† Achieves no crs overhead by additionally requiring random oracles.
* With the non-black box extractor, C∅C∅ retains the subversion resistance of the underlying SNARK [Bag19].
Thank you